ETDK-dolgozat

Jakab Gergő

Papp Gellért-Szabolcs

XXVIII. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció

Kolozsvár, 2025. május 15–18.

Watch My Project

Közösségi projektek állapotának publikus követése

Szerzők:

Jakab Gergő

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Papp Gellért-Szabolcs

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatikai matematika szak, III. év

Témavezetők:

dr. Simon Károly, egyetemi adjunktus,

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Balogh Szabolcs, szoftverfejlesztő,

Codespring

Hegedüs Hunor, szoftverfejlesztő,

Codespring

Watch My Project: Közösségi projektek állapotának publikus követése

Jakab Gergő, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év
Papp Gellért-Szabolcs, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatikai matematika szak, III. év

dr. Simon Károly, egyetemi adjunktus,
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Balogh Szabolcs, szoftverfejlesztő,
Codespring
Hegedüs Hunor, szoftverfejlesztő,
Codespring

A Watch My Project egy olyan szoftverrendszer, amely lehetővé teszi közösségi projektek állapotának publikus követését. Az alkalmazás teret ad civil szervezetek, egyesületek és intézmények számára ahhoz, hogy projekteket hozzanak létre, valamint ezekhez feladatokat és mérföldköveket határozzanak meg. Ezek révén átláthatóvá válnak a kezdeményezések megvalósításának feltételei és kivitelezési szakaszai.

A meghatározott fázisokhoz különböző állapotok rendelhetőek, illetve bejegyzések, kérdőívek is létrehozhatóak hozzájuk. A szervezetek így tájékoztathatják a közösségeket a folyamatban levő programjaik aktuális állapotáról, kezdve a tervezési fázistól, a kivitelezési folyamaton keresztül, egészen a projekt befejezéséig.

A felület lehetőséget teremt a létrehozott szervezetek és megvalósítások böngészésére, valamint a regisztrált felhasználóknak lehetőséget ad arra, hogy feliratkozzanak projektekre és naprakész tájékoztatást kapjanak ezekről, vagy akár be is kapcsolódhatnak ezek megvalósításába.

Tartalomjegyzék

Bevezető

Egy közösség számára fontos lehet a környezetükben végbemenő közösségi projektek céljainak, folyamatának és szükségleteinek követése. Ezeknek az információknak a birtokában könnyebben hozzá tudnak járulni a projektek haladásához, adományok vagy önkéntes segítség formájában. A civil szervezeteknek vagy intézményeknek is megéri ezeket az információkat megosztani a közösséggel, hogy ezáltal támogatáshoz jussanak.

Alapos utánajárást követően arra a következtetésre jutottunk, hogy jelenleg nincs túl sok modern, és kényelmes megoldás, amely a Watch My Project-hez hasonló felületet biztosít a közösségi projektek követésére. Az alternatívák közt szerepelnek újságok, híradók, rádió csatornák, online hírportálok vagy a közösségi projekteket szervező szervezetek saját weboldalai. Ezekkel kapcsolatban azt lehet megfigyelni, hogy nagyon helyhez kötöttek, nem egységesek. Ha több projekt állapotát is nyomon szeretnénk követni, akkor kénytelenek vagyunk több helyről információt szerezni.

Vannak olyan felületek is, amelyek a Watch My Project-hez hasonlóan összefogó jellegűek, több projektet tesznek nyilvánossá. Ilyen például a Fuss NEKI! és a Rotary Székelyudvarhely. Viszont ezek helyhez kötöttek, és csak limitált formában adnak információt a projektek aktuális állapotáról.

Mindezek alapján a Watch My Project célja egy felületet nyújtani a civil szervezetek, egyesületek és intézmények számára ahhoz, hogy megosszák a közösséggel a projektjeik céljait, lépéseit, szükségleteit. A felületen belül ki vannak emelve a projekteknek azok a fázisai, amelyekben a felhasználók direkt módon tudnak segíteni, például önkéntességgel vagy adományokkal. Így a közösség informálódik a környezetében végbemenő projektekről, illetve a projektek támogatókat és önkénteseket nyerhetnek. Mindez egy modern, letisztult megjelenéssel rendelkező weboldalon keresztül valósul meg, amely egyszerűen megérthető és kezelhető, mind a felhasználók, mind a szervezetek és intézmények számára.

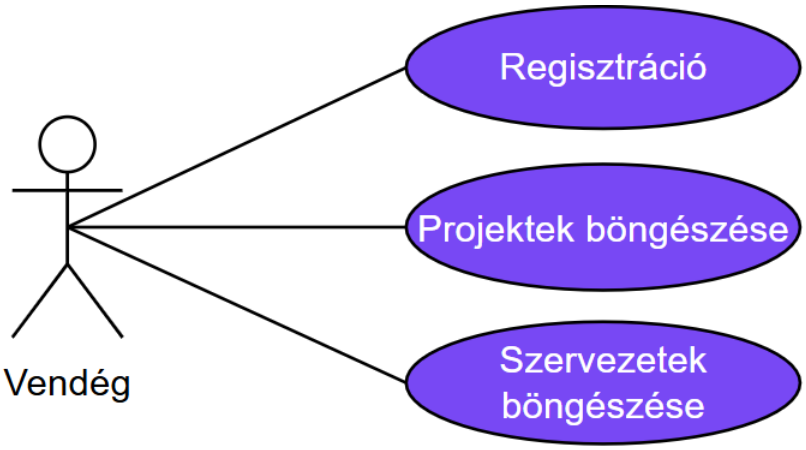
A dolgozat a következező fejezetekben bemutatja az alkalmazás fejlesztési folyamatát, felépítését, függőségeit és működését.

Az 1. fejezet, a projekt felhasználói szerepköreit és a szerepköröknek megfelelő szolgáltatásait foglalja magába. A 2. fejezet az alkalmazás szerkezetét mutatja be. Részletezi a szerver és a web alapú felhasználói felület működését, a technológiák használatát, a fontosabb konkrét implementációs megoldásokat és a kitelepítési folyamatot. A 3. fejezetben általános információk találhatóak a projekt fejlesztése során felhasznált fontosabb technológiákról és

eszközökről. A 4. fejezet az alkalmazás funkcionalitásait, az oldal működését mutatja be. Végül, a dolgozat utolsó fejezete összegez és továbbfejlesztési lehetőségeket sorol fel.

A projekt Székelyudvarhelyen, 2024 nyarán indult el, a Codespring és IT Plus mentorprogram keretein belül. A szakmai gyakorlat ideje alatt a jelen dolgozat szerzői Tőkés Róbert-Attilával közösen fejlesztették az alkalmazást. A fejlesztés folytatódott az egyetemi csoportos projekt tantárgyon belül, ahol egy egyetemi félév idejére a csapathoz csatlakozott Péter Csongor-Attila, Sorbán Előd és Szász Csongor.

A dolgozat írói köszönetet szeretnének mondani mentoraiknak, Balogh Szabolcsnak és Hegedüs Hunornak, szakmai koordinátoruknak, dr. Simon Károlynak, az fejlesztésbe ideiglenesen becsatlakozó kollégáknak, Péter Csongor-Attilának, Sorbán Előднek, Szász Csongornak, Tőkés Róbert-Attilának, illetve a Codespring csapatának a dolgozat és prototípus megvalósításához való hozzájárulásukért.



1. ábra. Vendég felhasználó funkcionalitásai

1. Szerepkörök és funkcionalitások

A rendszer négyféle szerepkört különböztet meg: nem regisztrált (vendég) felhasználó, regisztrált felhasználó, projekt szervező és adminisztrátor. A felhasználók a szerepkörüknek megfelelő jogosultságok mentén érhetik el az alkalmazás funkcióit.

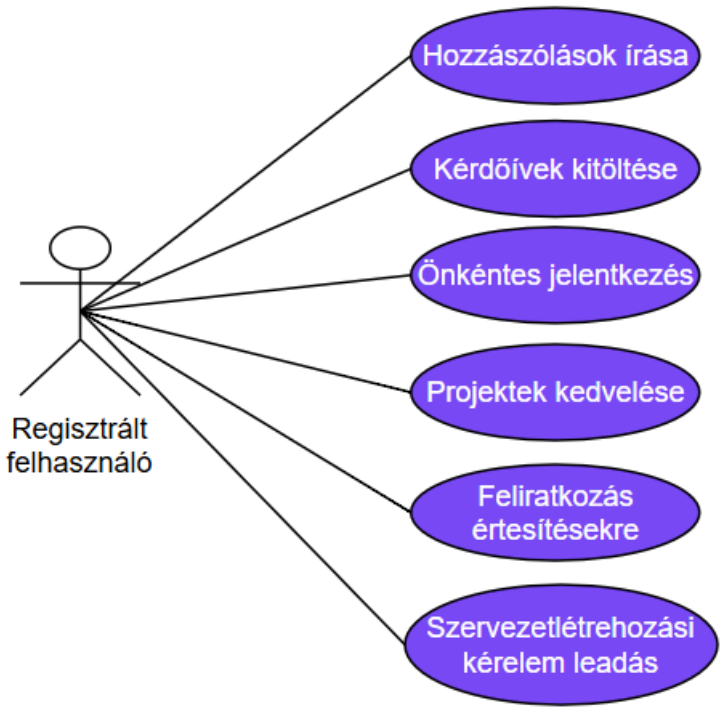
Bejelentkezés után a rendszer a böngészőben tárolja a felhasználó azonosításához szükséges információkat. Ha ezek az adatok nem érhetőek el, akkor az oldalon kevesebb művelet hajtható végre és bizonyos gombok és menüpontok elérhetetlené vagy láthatatlanná válnak.

1.1. Nem regisztrált (vendég) felhasználó

Az alkalmazás alap megjelenítése a vendég felhasználók nézete, amely lényegében csak olvasási műveletekhez való jogot jelent. Az 1. ábrán láthatóak az így elérhető funkcionalitások.

A nem regisztrált felhasználók böngészhetik a projekteket és szervezeteket, valamint megtekinthetik azok részleteit. Az egyes kezdeményezéseknél létrehozott kérdőívek és bejegyzések is elérhetőek, de ezekkel nem lehet semmilyen interakcióba lépni.

Ahhoz, hogy az oldal több mozgásteret adjon egy felhasználónak, regisztráció szükséges. Regisztrációnál a Google által szolgáltatott bejelentkezést alkalmazva, a felhasználó választhat az eszközén használt e-mail címek közül vagy új e-mail címet megadva is regisztrálhat a rendszerbe.



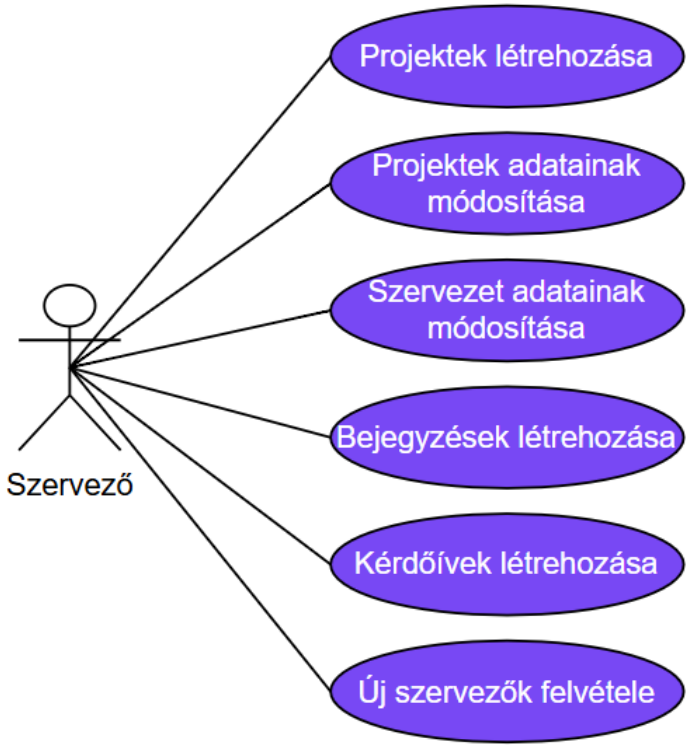
2. ábra. Regisztrált felhasználó funkcionalitásai

1.2. Regisztrált felhasználó

Regisztrált felhasználók számára kibővülnek a vendég szerepkörben említett funkcionalitások. Ahogy a 2. ábrán látható, lehetőség nyílik a projekteknél megadott bejegyzésekhez hozzászólni, szavazásokon, kérdőíveken részt venni, illetve önkéntesnek jelentkezni, azoknál a projekteknél, ahol a szervezetek lehetővé teszik a közösség ilyenfajta bekapcsolódását a megvalósításba.

Míg vendégek számára nem volt elérhető a projektet megjelenítő kártyákon, az értesítésekre való feliratkozás és a kedvelési ikonok, regisztráció után ezek is kattinthatóak lesznek. A felhasználók a csengő ikonra kattintva megtekinthetik a szervezetektől kapott értesítéseket, míg a csillag ikon segítségével gyorsan elérhetik a kedvencek közé tett projektjeiket.

A regisztrált felhasználók szervezetlétrehozási kérelmeket is benyújthatnak, amelyhez egy űrlapot kell kitölteniük. A leadás után az adminisztrátorok döntenek arról, hogy a szervezet bekerülhet-e a rendszerbe. A regisztrált felhasználó, elfogadott kérelem esetén, projekt szervezővé válik az új szervezeten belül.



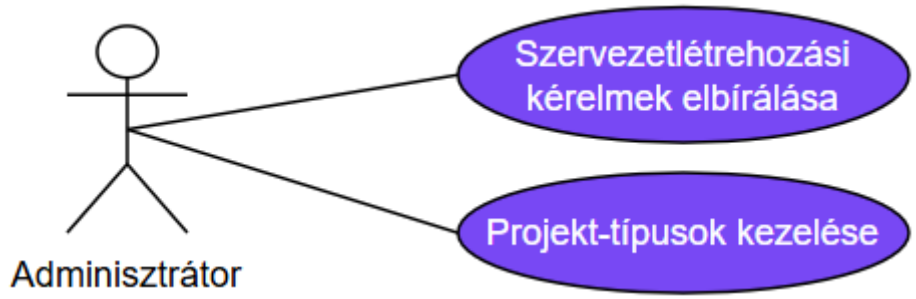
3. ábra. Szervező funkcionalitásai

1.3. Szervező

A szervezők olyan regisztrált felhasználók, akik egy vagy több szervezeten belül tevékenykednek, így módosíthatják ezek és az általuk létrehozott projektek információit (mérföldkövek, leírások stb.).

A szervezetlétrehozási kérelmek mellett egy másik módja is van annak, hogy egy felhasználó szervezővé váljon: ebben az esetben egy már meglévő szervezet küld szervezői meghívást a számára. Kezdetben csak a szervezet alapítója adhat hozzá új szervezőket a szervezethez, majd az így csatlakozott szervezők is jogosultak lesznek erre.

Ahogy a 3. ábrán is látható, a szervezők frissíthetik a szervezetükhöz megadott adatokat, valamint új projekteket hozhatnak létre és szerkeszthetik azok adatait. Emellett kérdőíveket és bejegyzéseket is tehetnek közzé a kezdeményezések oldalán. Az újonnan létrehozott projektek kezdetben csak a szervezők számára jelennek meg, azonban a szervezet oldalán lehetőség van állítani ezek láthatóságán a későbbiekben.



4. ábra. Adminisztrátor funkcionalitásai

1.4. Adminisztrátor

Az adminisztrátorok, olyan regisztrált felhasználók, akik a többi szerepkörhöz képest, a 4. ábrán feltüntetett további funkcionalitásokkal rendelkeznek.

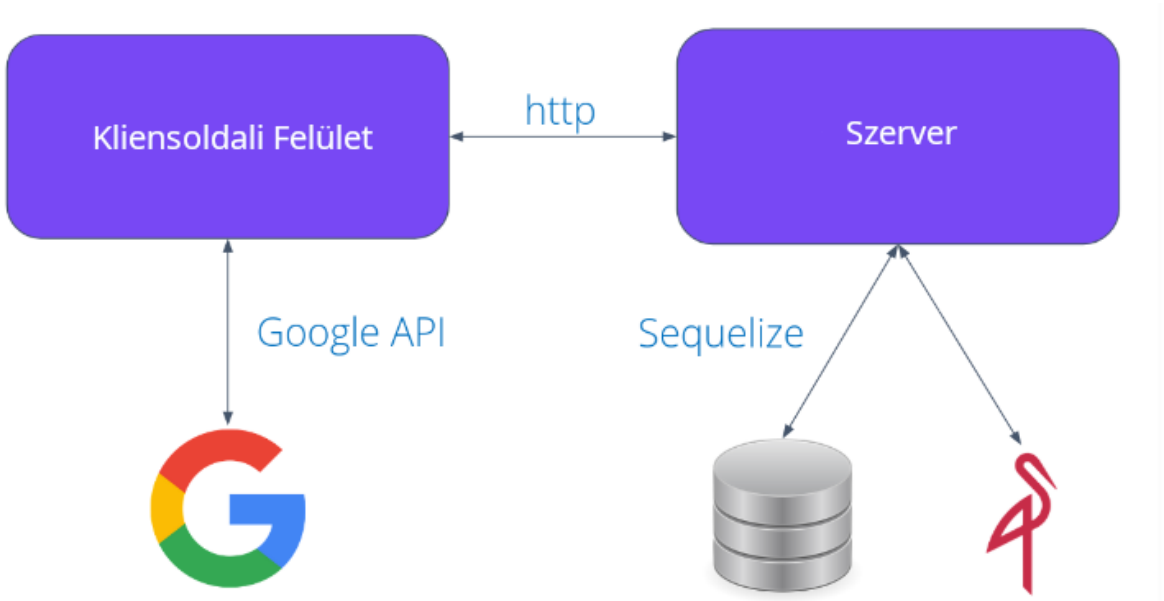
Lehetőségük van módosítani a kezdeményezésekhez rendelhető projekt-típusokat, amelyek alapján a létrehozott projektek is szűrhetőek. Emellett az ő feladatuk a szervezetlétrehozási kérelmek elbírálása is. Az adminisztrátorok megtekinthetik a leadott kérelmeket és eldönthetik, hogy a szervezet bekerüljön-e a rendszerbe.

Fontos megjegyezni, hogy az adminisztrátori szerepkör nem zárja ki azt, hogy az adott felhasználó szervezőként is tevékenykedhessen egy szervezeten belül.

2. Szerkezet

A szoftverrendszer két fő komponensből áll, ezek a szerver és a kliensoldali felület, amelyek elvégzik az üzleti logikával kapcsolatos műveleteket és az adatok megjelenítését. Ehhez a kettőhöz csatlakozik az adattárolási réteg, ami egy adatbázisból és média tárolóból áll. Ez a szerkezet szemléltetve van az 5. ábrán.

- A kliensoldali felület célja, hogy a felhasználóknak egy modern megjelenésű és könnyen megérthető felületet biztosítson, a projektek és szervezetek megtekintésére, létrehozására, változtatására. A felhasználói interakciók hatására a felület HTTP kéréseken keresztül lép kapcsolatba a szerver api-al, ami elvégzi a kért műveleteket. Ezen kívül a Google szolgáltatásain keresztül hitelesítő kulcsokat ad a felhasználóknak.
- A szerverrel REST API-on [20] keresztül lehet kapcsolatba lépni. A kérések alapján lekéri vagy változtatja az adatbázisban, illetve a média tárolóban eltárolt adatokat és megfelelő választ küld vissza.
- A média tároló szerepe a különböző projektekhez hozzárendelt képek biztonságos és hatékony tárolása, illetve elérése.
- Az adatbázis biztosítja minden adat hatékony tárolását, gyors elérését.



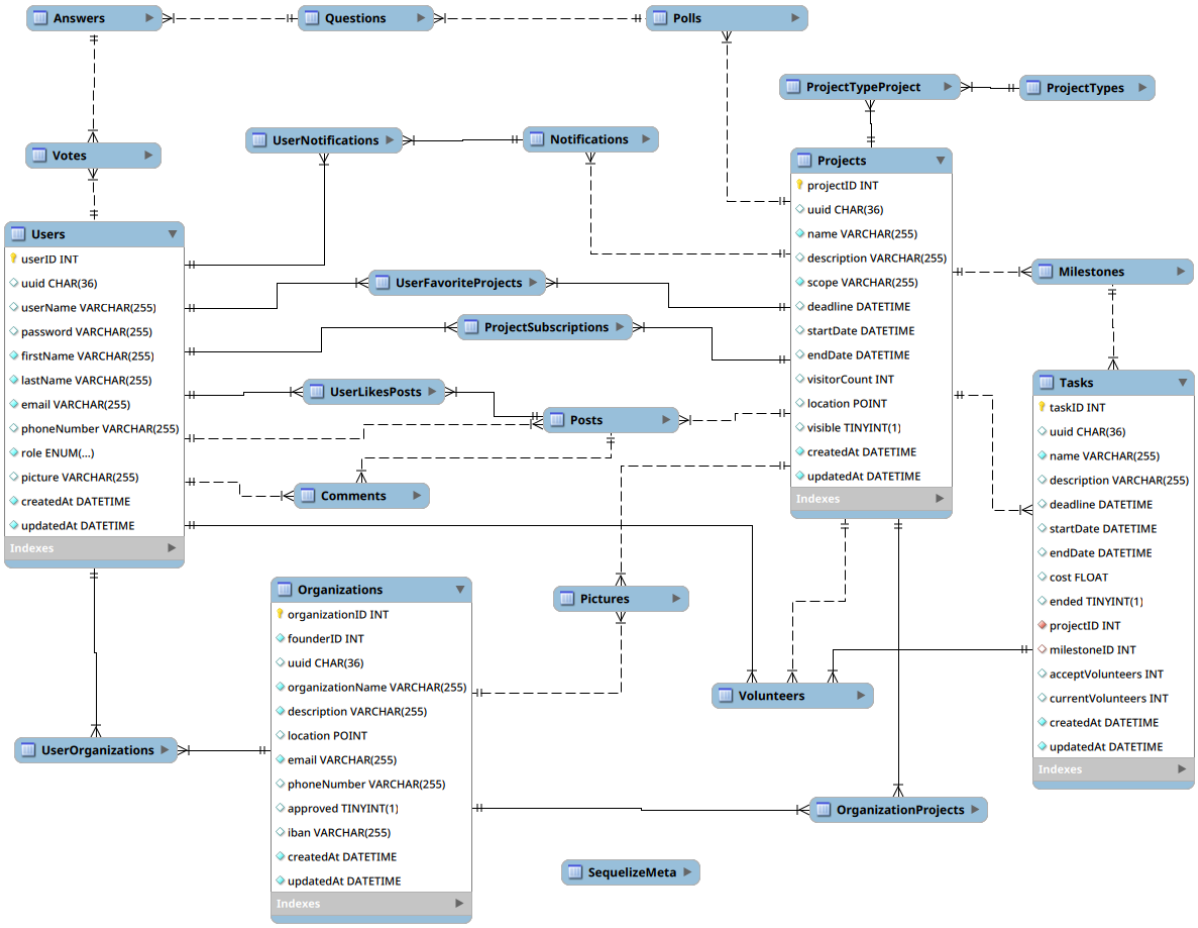
5. ábra. A Watch My Project szerkezete

2.1. Szerver

A szerver köti össze a projekt fő részeit, a felhasználói felületet az adatbázissal és a média tárolóval. Ezen kívül ez végzi el a perzisztens adatok kezelésével, valamint az üzleti logikával kapcsolatos műveleteket. A szerver JavaScript-ben, Node.js-el van implementálva

2.1.1. Adatmodell

Az adatok MySQL [2] adatbázisban vannak eltárolva. Jelenleg 15 entitást tart számon a rendszer és a közöttük levő kapcsolatokat. Az adatbázis legfontosabb entitásai a *felhasználók, szervezetek, projektek és feladatok*. Ez a négy entitás, az adatbázis szintjén, majdnem minden más entitással direkt kapcsolatban van, amint azt szemlélteti a 6. ábra.



6. ábra. Az adatbázis szerkezete

Az adatmodell létrehozása, karbantartása és az adatok elérése a Sequelize [3] ORM (Object-relational mapping) keretrendszer segítségével van megvalósítva. **Az adatbázishoz való kapcsolódást** a Sequelize-zal egyszerűen meg lehet oldani, ezután a *sync* függvénnyel

az új sémákat létre lehet hozni. Az 1. kódrészlet erre ad példát.

```
const dbConnection = new Sequelize(process.env.DB_NAME,
  process.env.MYSQL_USER, process.env.MYSQL_PASSWORD, {
    host: process.env.MYSQL_HOST,
    dialect: "mysql",
});
dbConnection
  .authenticate()
  .then(() => console.log("Database connected..."))
  .catch((err) => console.log("Error: " + err));
await dbConnection.sync();
```

1. kódrészlet. Adatbázishoz való kapcsolódás, és új sémák létrehozása

Az adatbázis sémákat is a Sequelize-al lehet definiálni. Ezt a *define* függvénnyel lehet elérni, ami paraméterként megkapja a tábla nevet, és egy JSON objektumot, ami leírja a tábla oszlopainak a nevét és azoknak a tulajdonságait. Ezen kívül, két tábla közötti kapcsolat létrehozására ad lehetőséget a *hasMany*, *belongsTo*, *belongsToMany* függvényekkel. **Az adatbázisban található adatok eléréséhez** a Sequelize minden táblához megad függvényeket, mint *find*, *findById*, *insert*, amelyek ezekhez biztosítanak hozzáférési lehetőséget.

A bejövő és kimenő adatokat a szerver több helyen DTO (Data Transfer Object) formájában reprezentálja, így a művelettől függően ugyanazt az entitást több vagy kevesebb attribútummal tartja számon. Így biztosítható az adatok biztonsága és a kommunikáció hatékonysága.

2.1.2. REST API

A szerver egy REST (REpresentational State Transfer) API-on [20] keresztül érhető el a kliensek számára. Az API http kéréseket fogad, amelyek vonatkozhatnak adatok lekérésére, változtatására, létrehozására, törlésére vagy a felhasználók hitelesítésére. Az így alkotott rendszer egyszerűen használható kliens oldalról, és egyszerűen fejleszthető szerver oldalon, hiszen a különböző API végpontok egymástól függetlenek, és minden kérés önmagában tartalmazza az összes szükséges információt. Az API-ra küldött kérésekre a státusz kód mellett JSON formátumú adatok érkezhetnek válaszként.

Az API implementálásához Express.js [18] volt használva, ami a REST API írásához ad keretrendszert.

Egy REST API-t szolgáltató szervert az *express* függvénnyel lehet létrehozni és a *listen* függvénnyel elindítani. Ehhez több router rendelhető, amelyek mindegyike kezelője minden

olyan http kérésnek, ami egy bizonyos prefixummal rendelkező végpontra fog érkezni. Ezzel a fajta megvalósítással modularizálhatóak a különféle kérések, tehát sokkal áttekinthetőbb lesz a kód. Az alkalmazás több ilyen routerrel rendelkezik, például: commentRouter, projectRouter, organizationRouter, amelyek rendre a kommentek, projektek, szervezetekhez tartozó http kéréseket kezelik le.

Egy router úgy működik, hogy az első paraméterként megadott végpontra érkezett kérés esetén, a kérést, egy válasz objektumot és egy olyan függvényt ad a második paraméterként megadott middleware-nek, amellyel ez át tudja adni ezt a három paramétert a következő paraméterként megadott middleware-nek. A router egyszerre több ilyen végpontot is tud kezelni, továbbá a GET kéréseken kívül képes a POST, PUT, PATCH, DELETE kérésekre is válaszolni a megfelelő függvény használatával.

A szerver szerkezete a többrétegű architektúra felépítését követi, tehát a különböző fajta feladatokat, a program különböző rétegei hajtják végre mielőtt átadnák a következő rétegnek az irányítást.

A szerver rétegei:

- **Router**: Ez a réteg felelős azért, hogy fogadja a felhasználók által küldött http kéréseket, és meghívja azokat a middleware-eket, amelyek megvizsgálják a kérések formátumának helyességét és a felhasználó hitelesítő adatait. Ha egy kérés helyesnek bizonyul, akkor az útválasztó meghívja a kontroller réteg megfelelő függvényét.

- **Kontroller**: A kontroller réteg feldolgozza a kérések tartalmát, a feldogozott adatokkal meghívja a szolgáltatás réteg megfelelő függvényeit, és az ezek által visszatérített eredmények alapján válaszol a kliensnek.

- **Szolgáltatás**: Ennek a rétegnek az a szerepe, hogy elvégezze a kérések kiszolgálásához szükséges műveleteket, illetve az adatok kezeléséhez szükséges CRUD (Create, Read, Update, Delete) műveleteket. Ezt úgy valósítja meg, hogy kapcsolatba lép az adatbázissal a Sequelize-on keresztül.

2.1.3. Képek feltöltése

Ha egy felhasználónak szervezői joga van egy projektre, akkor hozzá tud adni képeket a projekt profiljához. Képek feltöltéséhez a kliensnek POST kérést kell küldenie az /api/storage/upload/image(s) végpontra, amely tartalmazza a feltöltendő állományt vagy

állományokat bináris formában, a hozzátartozó adataikkal együtt. Miután a rendszer megbizonyosodott, hogy az állomány mérete nem túl nagy, beteszi azt a média tárolóba.

A képek tárolását egy MinIO szerverrel [22] valósítja meg az alkalmazás. A MinIO JavaScript könyvtárának segítségével kapcsolódik a Watch My Project szerver, a futó MinIO szerverhez, ezen a MinIO kliensen keresztül lehet interakcióba lépni a média tárolóval. Például, a képfeltöltés megoldható a kliens *putObject* függvényével, a törlés pedig a *removeObject* függvényével.

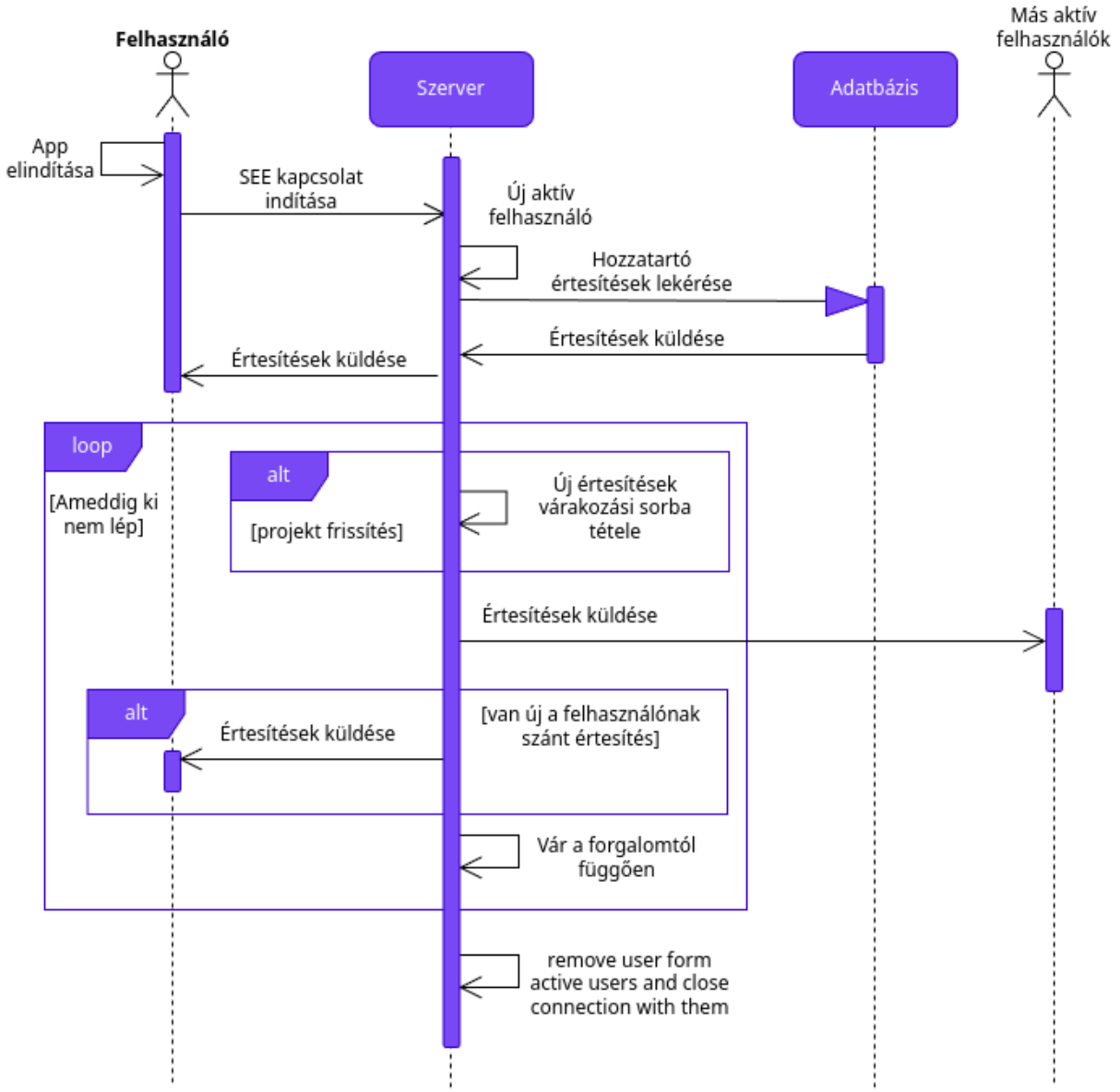
2.1.4. Értesítések küldése

A szerver az értesítések küldését kétféle módon valósítja meg:

- Egy bejelentkezett felhasználó a jelenlegi értesítéseit lekérheti direkt módon, ha GET kérést küld az /api/notifications végpontra.
- Alternatív módon SSE (Server Sent Event) [29] technológiával, a felhasználó feliratkozhat arra, hogy mindig, amikor új üzenete van a szerver oldalon, azt a szerver automatikusan elküldje neki. Az SSE hasonlít a web socket [5] protokollhoz, ugyanúgy kliens-szerver kapcsolatot alakít ki, viszont míg a web socket protokoll két irányú kommunikációt valósít meg, az SSE csak a szerver üzeneteit továbbítja a kliensnek. Az SSE használata megfelel az értesítések küldésére, ugyanis itt nincs szükség a felhasználó visszajelzésére. Így hatékony módon elérhető, hogy valós időben megkaphassák a kliensek a nekik szánt értesítéseket.

A valós idejű értesítések küldésének menete a következő: először a kliens küld egy GET kérést a szerver /api/notifications/subscribeToNotifier végpontjára, aminek hatására a szerver először megbizonyosodik arról, hogy a felhasználó nem kap értesítéseket egy másik ilyen kapcsolaton keresztül, majd elküldi a kliensnek a jelenlegi összes olvasatlan értesítést, és elkezdi az új értesítések küldését, mindig mikor egy új jön létre.

A szerver számontartja az összes értesítésekre feliratkozott felhasználót egy halmazban, annak érdekében, hogy ne tudjon ugyanaz a felhasználó több mint egyszer feliratkozni. Ezen kívül, számontartja az összes ilyen felhasználó esetében, hogy milyen új értesítései vannak, amelyek nincsenek még elküldve. Rövid időközönként minden ilyen felhasználóra megvizsgálja, hogy érkezett-e új értesítése, ha igen, akkor elküldi azokat, vagy üres üzenetet

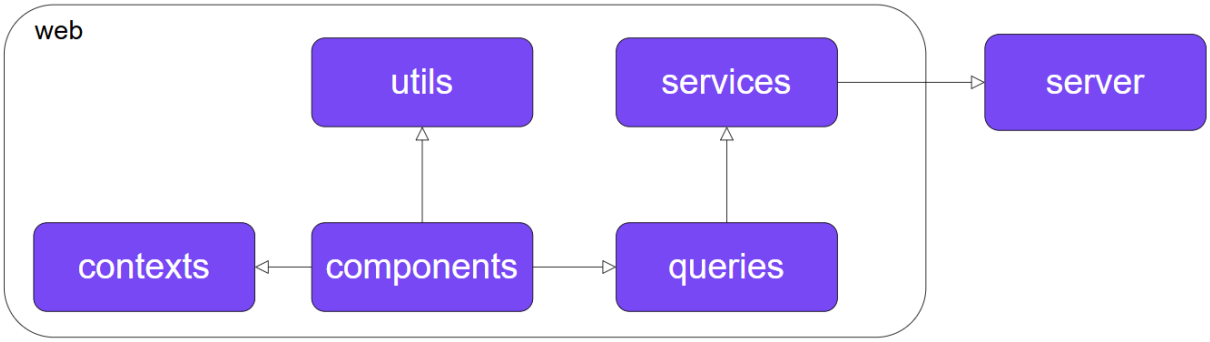


7. ábra. Szekvenciadiagram, ami leírja az értesítések küldésének menetét

küld annak érdekében, hogy ne záródjon be a kapcsolat a felhasználóval. Ennek működését szemlélteti a 7. ábrán látható szekvenciadiagram.

2.1.5. Autentikáció és autorizáció

Annak érdekében, hogy bejelentkezhessen a Watch My Project szolgáltatásba, a felhasználónak küldenie kell egy GET kérést a szerver /api/auth/google/login útvonalára, ahol a kérés body részében kell legyen egy Google által szolgáltatott hitelesítő JWT [17] (JSON web token). Ha a token helyes, akkor a kliens cookie-ban visszakap egy tokent, amivel a későbbiekben igazolja személyazonosságát, és ennek megfelelő jogosultságait. A token hitelesítése a Google által szolgáltatott OAuth2 [23] klienssel van megvalósítva.



8. ábra. Webes felület szerkezeti felépítése

A szerver attól függően, hogy a kliens milyen erőforráshoz próbál hozzáférni, elvárhatja tőle, hogy az 1. szekcióban említett szerepkörök közül rendelkezzen valamely szerepkörrel, aminek hiányában visszautasítja a kérést. Ezeknek a jogoknak az ellenőrzése az első dolog, amit a szerver elvégez, miután megnézi a kérés formájának a helyességét. A jogok vizsgálata middleware-eken keresztül történik, amelyek vizsgálják a felhasználó bejelentkezési státuszát és, ha szükséges, az adatbázison keresztül bizonyosodnak meg a felhasználó jogairól.

2.2. Webes felület

A webes felület a szoftverrendszer azon architektúrai eleme, amely a felhasználók számára is látható és általuk közvetlenül kezelhető. Ez a felület ReactJS segítségével készült, amelyről bővebben a 3.2. fejezetben esik szó.

2.2.1. Szerkezeti felépítés

A szerverhez hasonlóan a webes felület is JavaScript [7] alapú, és a rendszerhez illő React mappastruktúrában [8] került kialakításra.

A komponensek megfelelően strukturált mappákban találhatóak. Az első szintű felosztás a statikus és dinamikus fájlok között történik. Külön mappában találhatóak a webes felület állandó képei (logók, alapértelmezett képek) és az előre megadott szöveg állományok, amelyek többnyelvűségéről ugyancsak a 3.2. webes technológiákról szóló fejezetben található bővebb információ.

A dinamikus fájlok a konvencionálisan *src*-nek nevezett könyvtárban helyezkednek el. Itt több általánosan használt React fájlstruktúrát figyelembe véve, a szoftverrendszerhez legjobban találó szerkezetbe vannak csoportosítva a fájlok. Az elrendezés legfőbb mappái:

- **components**: Ebben a mappában találhatóak a webes felület önálló komponensei. Minden fájl egy-egy különálló felületi elemet tartalmaz. Az itt található fájlokat lehet a webes architektúra magjának is nevezni, ugyanis a többi mappa kódjai is ezeken a komponenseken keresztül kerülnek meghívásra.

- **contexts**: Ezek a fájlok az alkalmazás komponensei számára biztosítanak adatokat. Olyan információkat szolgáltatnak, amelyekre több helyen van szükség, mint például felhasználó információk, értesítésekkel kapcsolatos információk. Ezek az információk dinamikusan befolyásolják az oldal megjelenítését.

- **queries**: Egy másik különálló mappa az oldalon megjelenő adatok gyorsítótárazására (cache-elesére) szolgál. Ez a technológia csökkenti a szerverhez küldött kérések számát, ezzel is gyorsítva az adatbetöltést és javítva a felhasználói élményt.

- **services**: Ahogy a 8. ábrán is látható, ebben a mappában találhatóak a szerverrel történő kommunikációt megvalósító fájlok. Működésükről a következő alfejezetben esik szó.

- **utils**: Itt találhatóak az alkalmazás több pontján is felhasználható segédfüggvények. Például, egy dátumok megjelenítésére alkalmas átalakító függvény, ami a szervertől kapott dátumokat alakítja át a weben megjelenő formátumra.

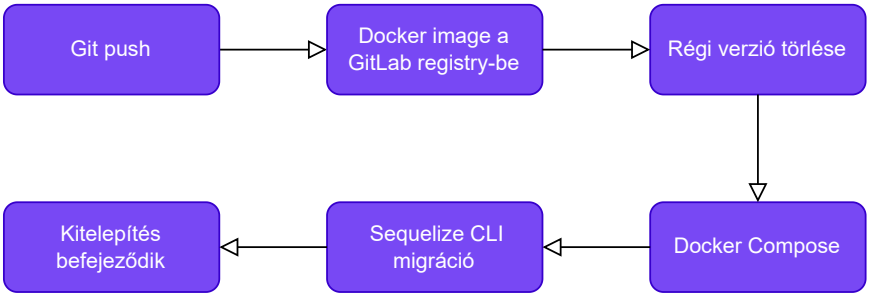
Az összes webes komponens által használt technológia részletesen van tárgyalva a dolgozat 3.2. fejezetében.

2.2.2. Szerverrel való kommunikáció

A szerver és a webes felület közötti kapcsolat a korábban említett service mappában található fájlok segítségével valósul meg. Az itt található fájlok mindegyike egy-egy, a 2.1.1 fejezetben tárgyalt entitáshoz kapcsolódik. Minden ilyen állomány tartalmazza az adott entitásokhoz tartozó alapvető HTTP-hívásokat (GET, POST, PATCH, DELETE) és egyéb szükséges metódusokat.

Az átláthatóság érdekében az API végpontok (endpointok) egységesen, REST API konvenciók [20] szerint vannak kialakítva. A kommunikáció során az adatok JSON-objektumok formájában mozognak a szerver és webes felület között.

A szerver felé küldött kérések Fetch API [4] segítségével történnek. Ez egy modern JavaScript interfész, amely számos konfigurációs lehetőséget biztosít a kérések beállításához,



9. ábra. A Watch My Project kitelepítése

például az endpoint URL, a header-ben található információk, a paraméterek stb.

A *fetch* függvény lehetővé teszi aszinkron kérések küldését a szerver felé. Meghívása után egy Promise [24] objektumot ad vissza, amelyet kiértékelve a webes felület a szervertől kapott válasznak megfelelően reagál.

2.3. Kitelepítés

A szerver és a kliensoldali felület ki van telepítve, és elérhető a *https://kozossegi-projektek-api.test.edu.codespring.ro* és *https://kozossegi-projektek.test.edu.codespring.ro* url-eken. Ezeket egy Traefik-es [30] reverse proxy biztosítja. A kitelepítés folyamatát szemlélteti a 9. ábra, ennek lépéseit a következő alfejezetek részletezik.

2.3.1. Gitlab CI

A szerver és a weboldal verziókövetéséhez GitLab [10] volt használva, mindkettő külön-külön Git tárolókba került. A projekt kitelepítéséhez a GitLab által szolgáltatott GitLab CI/CD [1] környezet került alkalmazásra, ami minden GitLab-bel való szinkronizálás (push) után elvégez folyamatos integrációs és kitelepítési műveleteket. Hogy testre lehessen szabni a CI/CD folyamatot, egy GitLab-on követett projekt a gyökerében kell tartalmazzon egy *.gitlab-ci.yml* nevű fájlt, ami tartalmazni fogja a folyamat leírását.

Jelenlegi verziójában a szerver kitelepítése három fázisból áll: build, test és deployment.

- A build (felépítés) fázis letölti a szerver függőségeit, utána lefordítja a szerver kódját, és átadja a függőségeket más fázisoknak artifact-eken keresztül.
- A test fázis statikus kódellenőrzést hajt végre az ESLint [11] segítségével.

- A deployment (kitelepítés) fázis csak akkor fut le ha a *develop* ággal történik szinkronizálás. Először bejelentkezik a GitLab projekt CI registry-jébe, és ide feltölt, egy a projektről készült Docker image-et. Ezután Docker Compose-zal kitelepíti a projektet.

2.3.2. Docker és Docker Compose

A kitelepített projekt Docker konténerekben fut. Így elérhető, hogy az alkalmazás különböző részei különböző környezetekben futhassanak úgy, hogy egymással tudjanak kommunikálni, mindezt hatékony módon. A projekt esetében a szerver, kliensoldali felület, adatbázis és MinIO szerver, mind külön konténerekben futnak. A 2. kódrészlet a szerver Docker image-ének a dockerfile-ja, ami elindítja a szervert egy Node.js-el rendelkező környezetben. A frontend hasonló módon oldja meg a saját docker image-ének készítését.

```
FROM node:20-alpine
WORKDIR /server
COPY package.json .
RUN npm i
COPY . .
CMD ["node", "./server.js"]
```

2. kódrészlet. A szerver image-ének dockerfile-ja

A teljes alkalmazás összetételét a szerver gyökerében levő *compose.yaml* állomány írja le. Ez leírja a projekt összes komponensére, hogy milyen image-ből származik, a szükséges környezeti változókat, milyen portokat osztanak meg a többi konténerrel, milyen hálózatokhoz vannak kapcsolódva, és meta információkat, amit a GitLab pipeline felhasználhat.

2.3.3. Adatbázis migrációk

Az adatbázis séma változások esetében kihívást jelent, ha meg is akarjuk őrizni az eddig tárolt adatokat a kitelepített alkalmazásban. Ennek a megoldására szükség van adatbázis migrációkra. A fejlesztés során a Sequelize CLI [28] (command line interface) parancssori eszköz volt használva az adatbázis migrációkra.

Miután inicializálva és konfigurálva van egy projektre a Sequelize CLI, a migrációs műveleteket parancssorból lehet elvégezni. Ahhoz, hogy migrációkat lehessen végezni, létre kell hozni migrációs állományokat, amelyek mindegyike két függvényt kell implementáljon, az *up* és *down* függvényeket, ahol az *up* elvégzi a migrációt, a *down* visszaállítja az adatbázist

a migráció ellőtti állapotába. Az *npx sequelize-cli db:migrate* parancssor utasítás elvégzi a nem felhasznált migrációkat, míg az *npx sequelize-cli db:migrate:undo* az utolsó lefuttatott migrációt semlegesíti azzal, hogy elvégzi a *down* függvényt.

3. Technológiák és eszközök

Ebben a fejezetben az alkalmazott technológiák mellett bemutatásra kerülnek a munkafolyamat során használt eszközök és módszerek.

3.1. Szerver technológiák

A felhasználói felület és a szerver is JavaScriptben íródott [7], szerver oldalon a **Node.js** [19] futtatási környezet volt alkalmazva.

Az adatok tárolására **MySQL** [2] volt felhasználva, egy nyílt forráskódú relációs adatbázis kezelő rendszer. A MySQL adatbázis Node.js szerverrel való összekötéséhez a **Sequelize** [3] ORM keretrendszer volt használva.

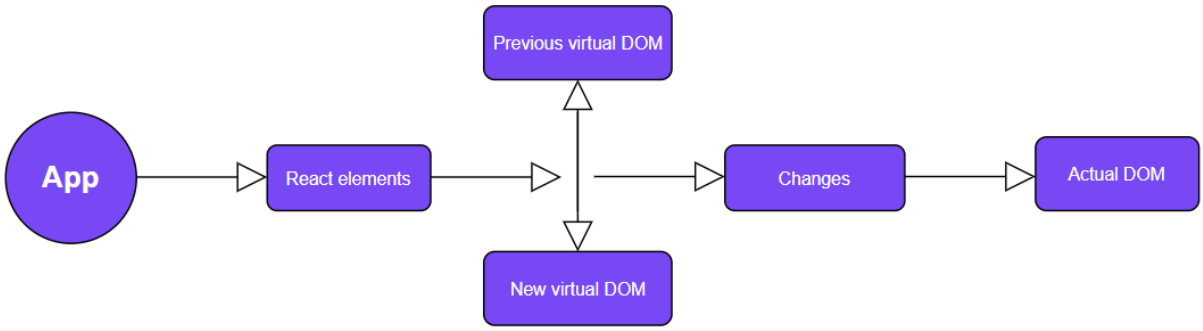
A képek tárolására a **MinIO Server** [22] volt igénybe véve, amely a Watch My Project szerver mellett lokálisan fut. A MinIO Server egy nagyon jól skálázható, magas teljesítményű objektum tároló rendszer.

3.2. Web technológiák

A webes felület esetében használt legfontosabb technológia a **React** [14], egy komponens alapú JavaScript könyvtár, amely JavaScript XML (JSX) szintaxist használ. Ennek segítségével HTML kódot lehet írni JavaScript-en belül. Fő célja az egyoldalas alkalmazások (single-page applications) fejlesztése, különös tekintettel a teljesítményre, a komponensek újrahasználhatóságára és az egyszerű karbantarthatóságára.

A React egy virtuális DOM-ot (Document Object Model) használ a weboldal felépítéséhez, és folyamatosan monitorizálja annak elemeit. Ennek köszönhetően, amikor változás történik az oldalon, a böngésző nem az egész DOM-ot rajzolja újra, hanem kizárólag a módosított komponenseket frissíti, így növelve a teljesítményt. Ezt a DOM manipulálási technikát a 10. ábra is szemlélteti.

A **TanStack Query** [16] egy olyan kliensoldali gyorsítótárazási (cache) rendszer, amely többféle frontend technológiához is alkalmazható. React projektek esetében a react-query csomagból importált *useQuery* hook segítségével valósítható meg az adatok cache-elése. Ezek a hook-ok a webes komponens és a szerver közötti *fetch* hívásokat csomagolják be. A query-k meghatározható ideig tárolják az aszinkron műveletek eredményeit. Amennyiben ezen az időtartamon belül ismét hívás történne ugyanarra az API-végpontra, a rendszer nem küld új



10. ábra. React működése

kérést a szerver felé, hanem az eltárolt adatot adja vissza. A cache tartalmának a lejárata vagy érvénytelenítése után azonban a webes felület által küldött kérések újra eljutnak a szerverig és onnan kapják meg az információkat.

A weboldalon működő nemzetköziesítés a **react-i18next** [15] keretrendszer alkalmazásával valósul meg. A különböző nyelvekhez tartozó fordításokat a megfelelő JSON fájlokból éri el a rendszer. Az oldal megnyitásakor a megjelenített nyelv automatikusan a felhasználó eszközén beállított nyelvhez igazodik. Amennyiben a weboldal nem rendelkezik fordítással az adott nyelven, akkor az alapértelmezett nyelv - az angol – kerül alkalmazásra. Az alkalmazás választható nyelvei: angol, magyar és román.

A szervezetlétrehozási űrlapnál használt **Leaflet** [12] egy nyílt forráskódú JavaScript könyvtár, amely interaktív térképek megjelenítésére szolgál. Az alkalmazásban a Leaflet által kínált funkciók közé tartozik a helységnév szerinti keresés, valamint térképen történő koordináta megjelölés.

A webes felület megjelenítését a **Material UI** (MUI) [13] tette egységessé. A MUI egy széles körben használt React-komponenskönyvtár, amely a Google Material Design irányelveit implementálja. Rengeteg előre definiált, könnyen személyre szabható komponenst tartalmaz, amelyek segítségével jó felhasználói élményt nyújtó webes felület hozható létre.

3.3. DevOps eszközök

A GitLab nyújt eszközöket folyamatos integrációra és kitelepítésre, a **GitLab CI/CD** által [1]. Ezzel a kódbázis integritását lehet biztosítani minden új verzió esetén.

A **Docker** [21] egy virtualizációs szoftver, amellyel megoldható pehelysúlyú operációs rendszerek és azokon a fejlesztés alatt levő program (részeinek) futtatása, minden szükséges függőséggel. A Docker virtualizációja (konténerek) abban különbözik a tradicionális virtuális

operációs rendszerektől, hogy felhasználja a host rendszer magját. Több konténerből álló rendszer orkesztrációjára használható a Docker Compose. A **Docker Compose**-zal [21] több konténert lehet elindítani és hálózati kapcsolását, viselkedését, portjait, meta adatait beállítani.

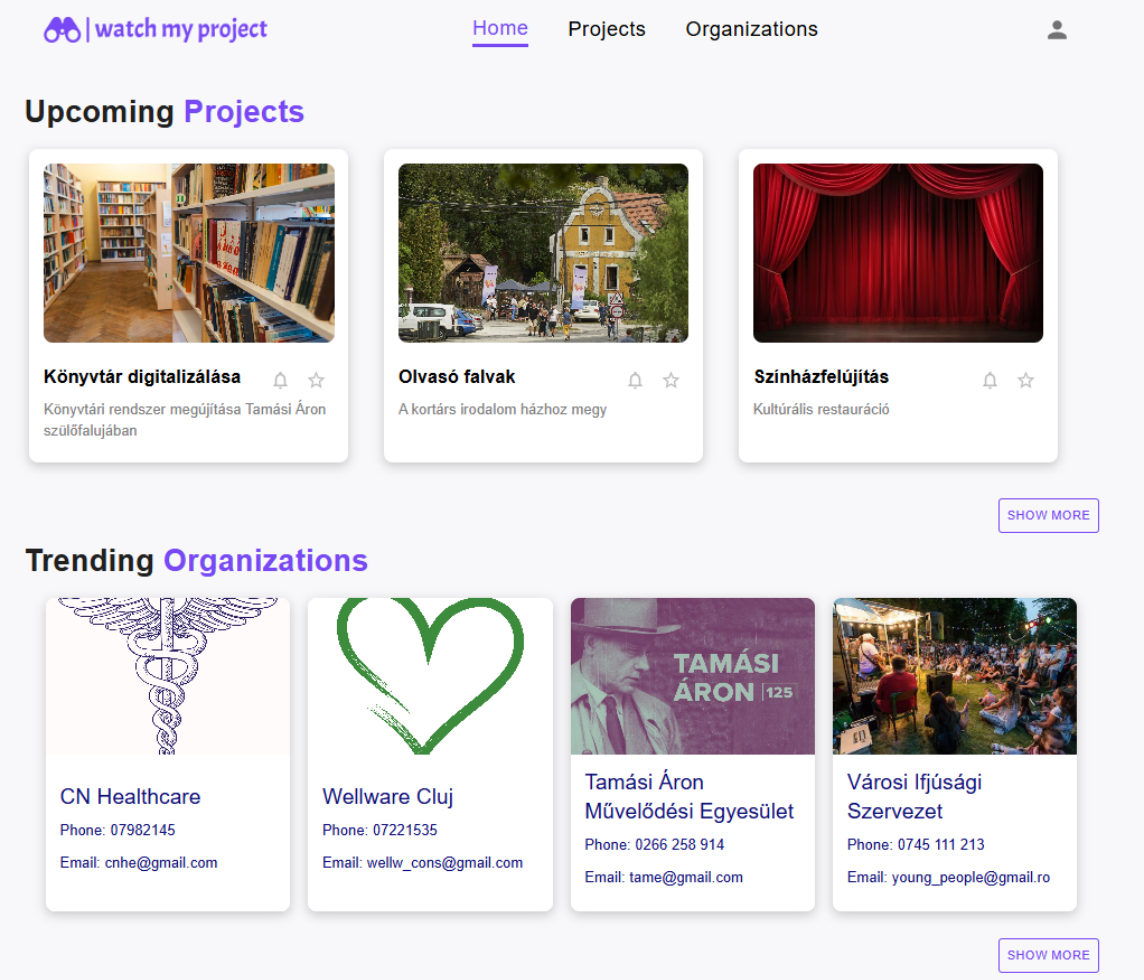
Az alkalmazás szerver és kliens oldali kódja egyaránt **ESLint** [11] kódelemző segítségével került karbantartásra. Az ESLint egy nyílt forráskódú JavaScript kódelemző (linter), amely lehetővé teszi a kódminőségét biztosító standardok és szabályok meghatározását. Ezeknek a szabályoknak a betartása folyamatosan ellenőrizhető a kód írása során, emellett a forráskód megfelelő formázása a kitelepítés során is alapvető követelmény: a kitelepítési pipeline csak akkor fut le sikeresen, ha a kód megfelel az előírt szabályoknak.

3.4. Fejlesztést elősegítő eszközök és módszerek

Verziókövetésre Git, illetve **GitLab** [10] volt használva. A GitLab egyik előnye az, hogy lehetséges lokális szerveren is futtatni.

A **Figma** [6] egy felhőalapú tervezőeszköz, amelyben a webes felület dizájnja volt megalkotva. A látványtervek mellett a Figma projektből könnyen kinyerhetőek voltak a meghatározott színkódok, ikonok és egyéb stilisztikai elemek is.

A fejlesztés során **Scrum** [27] munkamódszer volt alkalmazva. A Scrum metodológia alapján a munka 2-3 hetes fejlesztési szakaszokban, úgynevezett sprintekben zajlott. A rövid napi megbeszélések (daily) mellett, a sprintek között mindig sor került a befejezett sprint kiértékelésére és a következő fejlesztési ciklus megtervezésére.



11. ábra. A főoldal felépítése

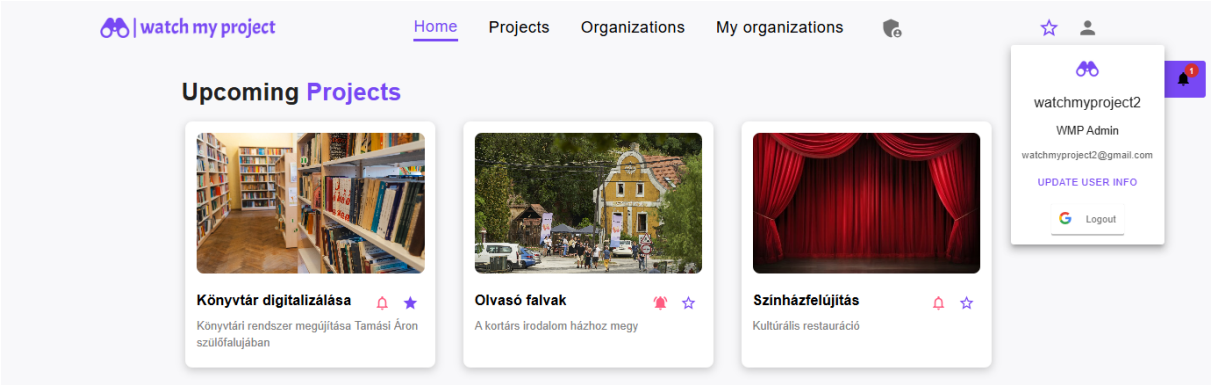
4. A szoftverrendszer működése

Az oldalak felépítése és a főoldal

Az alkalmazás oldalai a következőkben vázolt sablon alapján épülnek fel. Az oldal tetején található a navigációs sáv, ami az 1. fejezetben felsorolt felhasználói szerepköröknek megfelelően jeleníti meg az elérhető funkciókat.

Az oldal alján elhelyezkedő lábléc is tartalmazza a lényeges oldalakra mutató hivatkozásokat, valamint itt található a felület nyelvének kiválasztására szolgáló lehetőség is.

Az oldalak specifikus tartalma az említett két komponens között helyezkedik el. A 11. ábrán látható a főoldal felépítése, itt található néhány folyamatban lévő közösségi projekt, valamint néhány aktív szervezet. Az őket megjelenítő kártyákra kattintva, a kiválasztott elem részletes adatai tekinthetőek meg. A *Továbbiak megtekintése* gombokra kattintva pedig elérhetőek az oldalon létrehozott egyéb kezdeményezések vagy szervezetek.



12. ábra. Felhasználói adatok és bejelentkezés után elérhető funkciók

Bejelentkezés és felhasználói információk

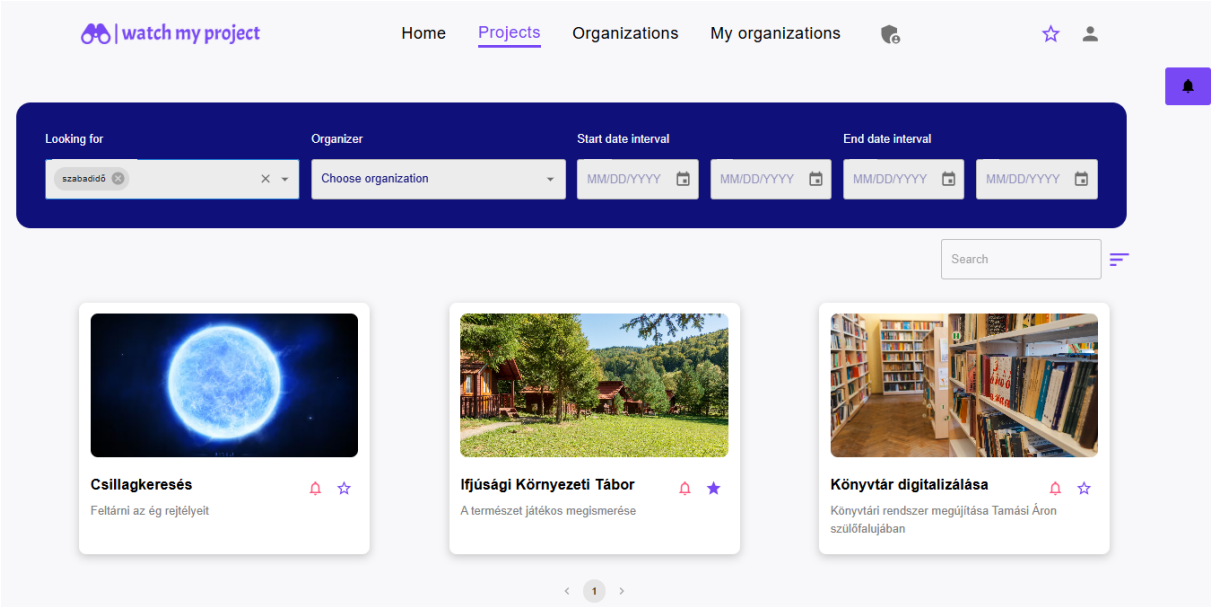
A be- és kijelentkezési folyamat a frontend-ről indul és a react-google-login csomag [26] komponenseinek használatával történik. Ezek a komponensek több konfigurálható attribútummal (konvencionális néven *props*) rendelkeznek, amelyek lehetővé teszik a Google-fiókok egyszerű integrálását saját rendszerekbe.

A bejelentkezés a navigációs sávban található felhasználó ikonon keresztül érhető el. Az ikon mögött található a Google autentikációs rendszerét használó hitelesítési gomb. A gombra kattintva egy új ablakban nyílik meg a Google fiókválasztási funkciója, ahol az eszközön elérhető Google-fiókok közül lehet választani vagy akár új, még nem használt e-mail cím is megadható a bejelentkezéshez. A fiók kiválasztását követően, a komponens felülírható attribútumainak köszönhetően, a Google-től kapott adatok továbbításra kerülnek a szerver felé. A szerver, a 2.1.5. fejezetben leírtaknak megfelelően feldolgozza ezeket, majd a visszakapott válasz után, az oldal a bejelentkezett felhasználó jogosultságainak megfelelően frissül.

Az aktuálisan bejelentkezett felhasználó adatait a rendszer *session storage*-ban tárolja és *useContext* hook [25] segítségével tartja nyilván a komponensekben, ennek köszönthetően az oldal elemei könnyen elrejthetőek a jogosultságoknak megfelelően.

Sikeres bejelentkezés után, ahogy a 12. ábra is szemlélteti, a felhasználó ikonra kattintva érhetőek el a felhasználói adatok, valamint a kijelentkezést szolgáló komponens, amely hasonló módon működik, mint a bejelentkezési komponens.

A rendszerben a felhasználó alapértelmezett adatai kezdetben megegyeznek a Google-fiókjához tartozó felhasználói információkkal (felhasználónév, vezetéknév, keresztnév). Azonban lehetőség van ezek módosítására és telefonszám megadására is, ha a felhasználó a fiókjától független adatokat szeretne használni az alkalmazásban.



13. ábra. Projektek böngészése

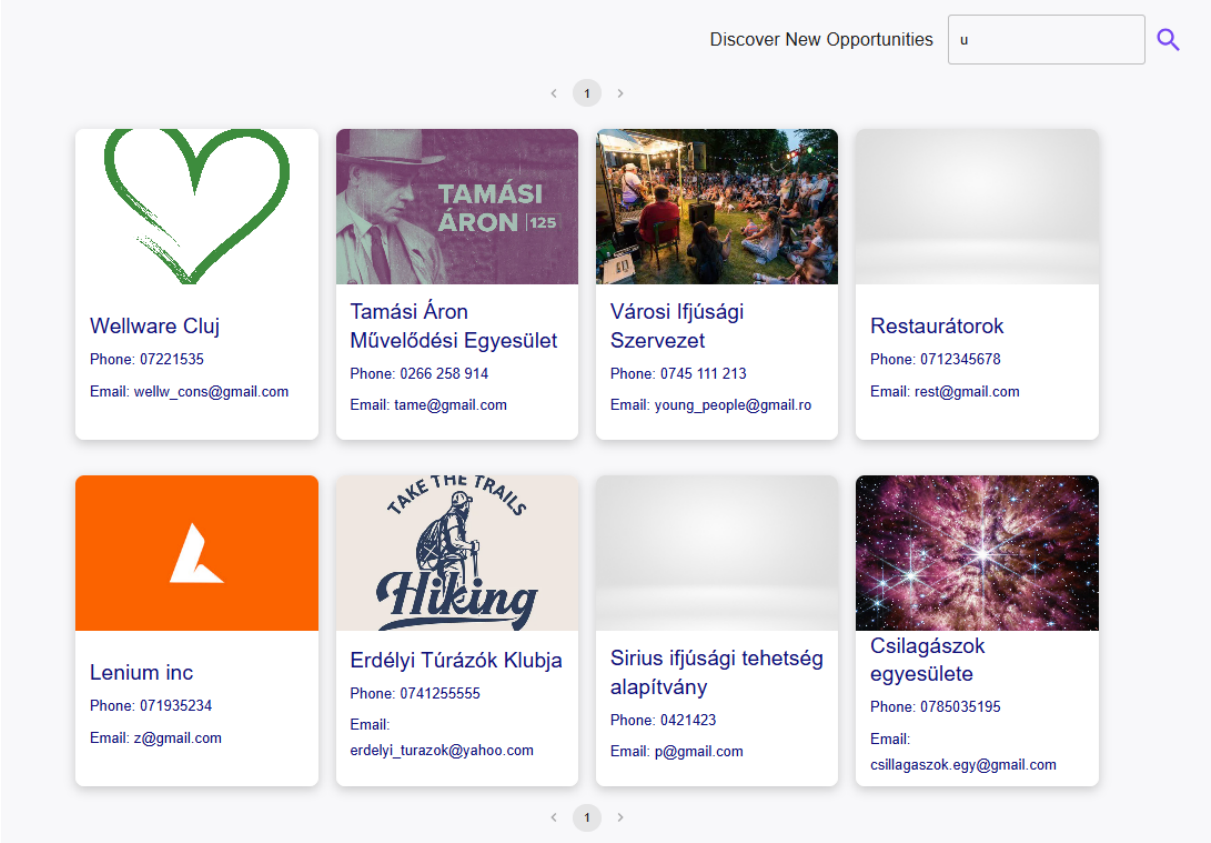
Szervezetek és projektek böngészése

Az alkalmazás központi elemeit a főoldalon is megjelenő kezdeményezések és szervezetek alkotják. Ezek kártyák formájában jelennek meg a hozzájuk tartozó legfontosabb információkkal és egy feltöltött képpel együtt. Azoknak a projekteknek és szervezeteknek az esetében, ahol a szervezők nem töltenek fel képet, alapértelmezettként egy szürke helytartó minta jelenik meg.

Bejelentkezés után a projektkártyákon megjelenik a csillag ikon, melynek segítségével a felhasználó kedvencek közé helyezhet projekteket. A kedvenc projektek böngészése a navigációs sávban található csillag ikonon keresztül lehetséges. A 2.1.4. fejezetben tárgyalt értesítésekre való feliratkozást, ugyancsak a projekt kártyákon elhelyezett csengő ikon által teheti meg a bejelentkezett felhasználó.

Amennyiben kizárólag projekteket, vagy szervezeteket szeretnénk böngészni, akkor többféle módon is átnavigálhatunk a megfelelő oldalra. Ez megtehető a navigációs sávon vagy az oldal alján a megfelelő menüpontot kiválasztva, illetve a kártyák alatt a *Továbbiak megtekintése* gombra kattintva.

A kiválasztott oldalon egyszerre több projekt vagy szervezet között lehet navigálni. A kezdeményezések böngészése során többféle szűrési feltétel is megadható (13. ábra). Beállítható a projekt-típus (például fesztivál, kultúra), valamint a kezdési és befejezési időintervallum. Emellett szűrés végezhető a projekteket létrehozó szervezetek vagy projektek



14. ábra. Szervezetek böngészése

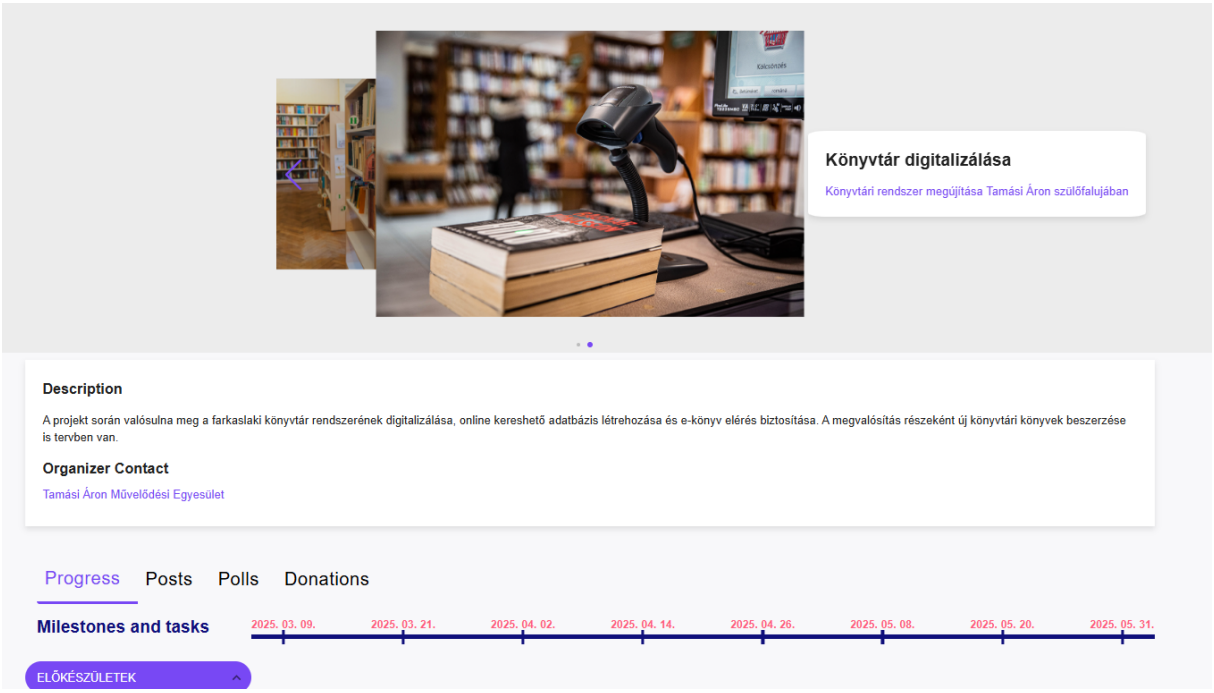
neve alapján is, valamint a kapott találatok ábécé- vagy időrendi sorrendben is rendezhetőek. A szervezetek oldalán név szerinti keresésre van lehetőség (14. ábra).

Projekt részleteinek megjelenítése

Egy projekt részletei (15. ábra) a hozzá tartozó kártyán keresztül érhetőek el. Egy kártyára kattintva az oldal tetején megjelennek a projekthez feltöltött képek (legtöbb három darab), valamint annak a neve és célja. Ezt követően megtalálható a részletesebb leírás és a projektet kezdeményező szervezet elérhetősége.

Az oldal további tartalma egy menüsor segítségével változtatható. A felhasználók négy interakcióra és tájékoztatásra alkalmas menü közül választhatnak: *Előrehaladás*, *Bejegyzések*, *Szavazások* és *Adományok*.

Az *Előrehaladás* alatt tekinthető meg a kezdeményezés ütemezési terve. Az ehhez létrehozott mérföldkövek és feladatok, 16. ábrán is látható, Gantt-diagramban való megjelenítése egy átlátható képet ad a megvalósítás teljes folyamatáról, valamint a feladatok egymástól való függőségéről, időrendiségéről. A feladatokhoz leírás, kezdési és befejezési



15. ábra. Projekt részletes oldala

időpont, valamint szükséges költségek is megadhatóak. Itt van megjelenítve az is, hogy a szervezetnek van-e szüksége önkéntesek bevonására egy adott feladathoz. A felhasználók időintervallumot is beállíthatnak, hogy csak egy kijelölt időszak mérföldkövei és feladatai jelenjenek meg a diagramban.

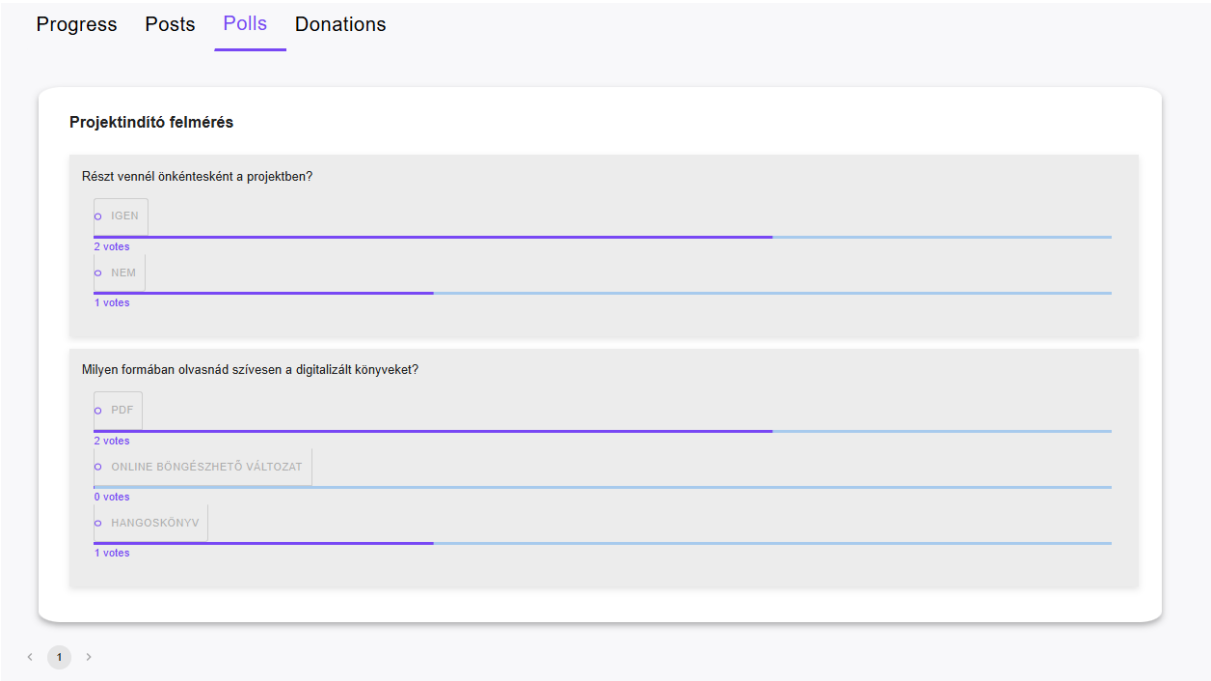
A *Bejegyzések* fül alatt láthatóak a projekthez kapcsolódó közlemények. A szervezetek itt közölhetnek friss információkat, amelyekhez a felhasználók hozzászólhatnak, valamint kedvelhetik az egyes bejegyzéseket.

A *Szavazások* menüpont alatt eldöntendő kérdéseket tartalmazó kérdőívek találhatóak. Minden kérdőívnek legalább egy kérdést és kérdésenként legalább két választási lehetőséget kell tartalmaznia. A kérdőívekhez beállítható egy időszak, ameddig azok kérdései elérhetőek és megszavazhatóak. Ahogy a 17. ábrán is látható, a szavazások eredményeit később a szervezők nyilvánossá tehetik.

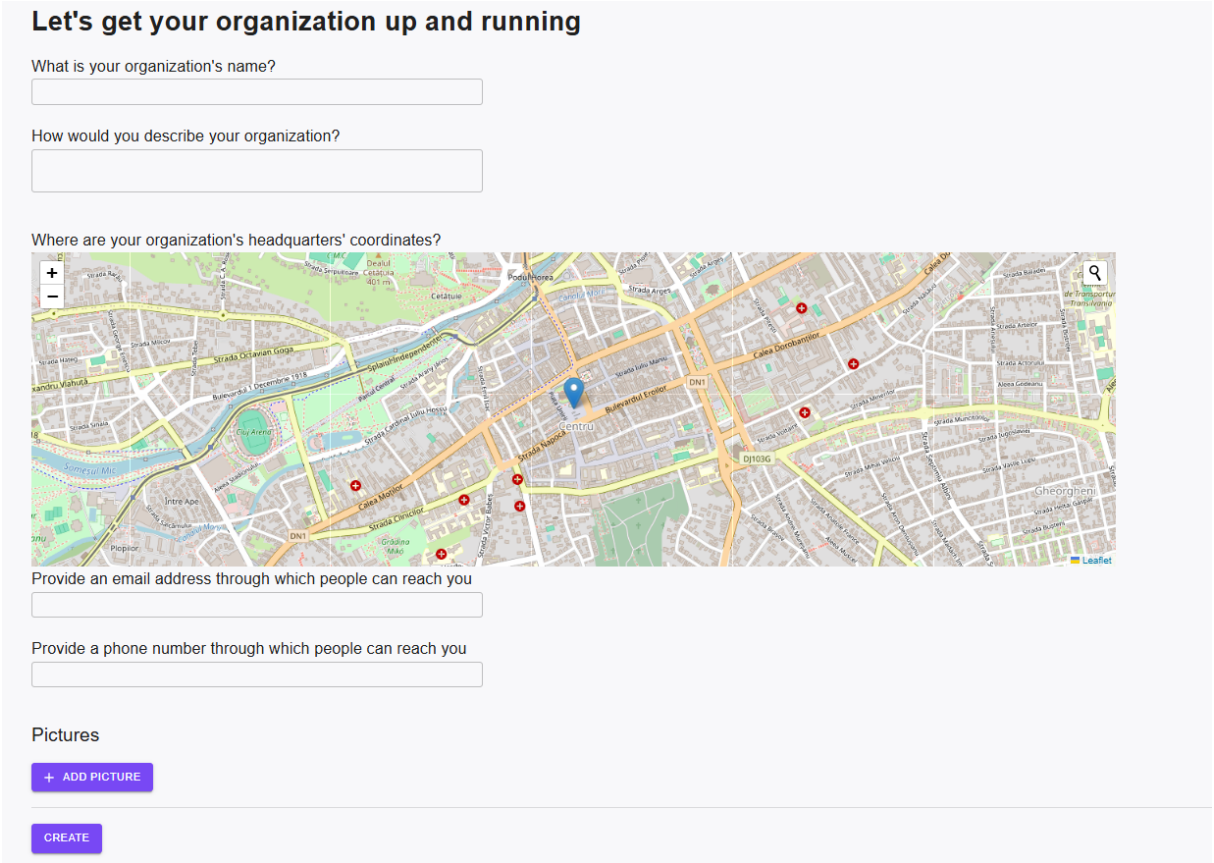
Az *Adományok* pont alatt a szervezetek feltüntethetnek támogatási elérhetőségeket, melyek segítségével az érdeklődők pénzbeli támogatást nyújthatnak a kezdeményezéseknek.

A szervezők jogosultak bejegyzéseket, kérdőíveket létrehozni, mérföldköveket, feladatokat módosítani. Ezek a létrehozási/módosítási űrlapok a megfelelő menüpontok alatt érhetőek el. Egy projekt részletes nézete szervezői szerepkörben még további olyan ikonokat tartalmaz, amelyek lehetőséget biztosítanak bejegyzések, kérdőívek törlésére, illetve projekt

16. ábra. Projekt előrehaladása



17. ábra. Szavazások



18. ábra. Szervezetlétrehozási űrlap

információinak gyors módosítására. A rendszer minden írási művelet esetén ellenőrzi a kapott bemenetet a hibák és helytelen adatok elkerülése érdekében.

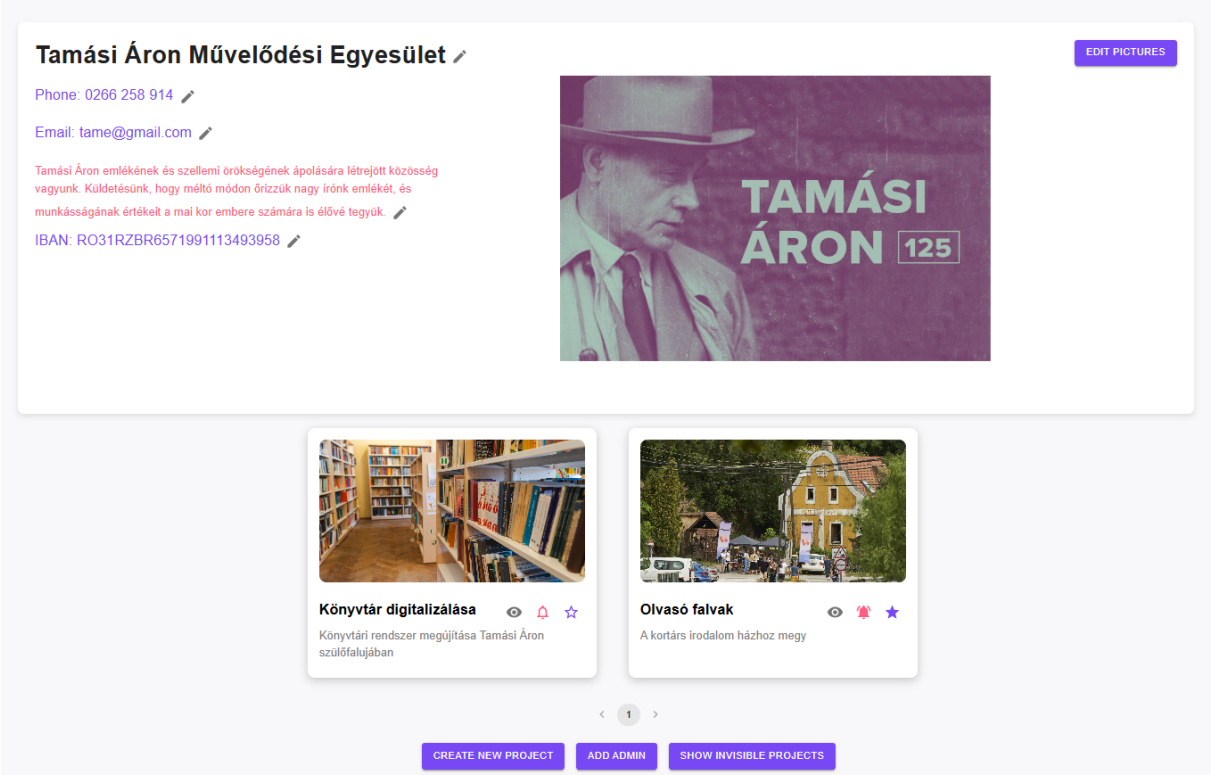
Szervezetlétrehozási űrlap

Projektek létrehozásához és kezeléséhez elengedhetetlen egy szervezet regisztrálása. Ezt a létrehozási kérelmet bármely bejelentkezett felhasználó benyújthatja. A szervezet regisztrációs űrlap az oldal alján található gombra kattintva érhető el.

Az űrlap a 18. ábrán megjelenő adatokat kéri a felhasználótól: szervezet neve, leírása, e-mail címe, telefonszáma és székhelyének koordinátái. Opcionálisan képek is csatolhatóak a kérelemhez.

A szervezet székhelyét a rendszer egy interaktív térképen elhelyezett jelölő (pin) alapján határozza meg, Geocoder [9] segítségével. A rendszer a térképen történő kijelölés mellett, a keresési funkciót is használja, amely segít a felhasználóknak a szervezet székhelyének pontosabb és gyorsabb meghatározásában.

A szervezetlétrehozási kérelmeket az adminisztrátorok érhetik el, ezek elbírálásával



19. ábra. Saját szervezet részletes oldala

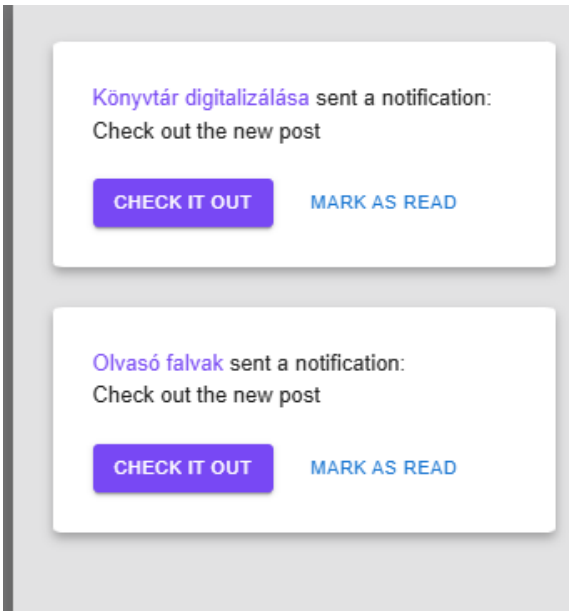
kapcsolatos lehetőségek az adminisztrátor funkcióknál találhatóak.

Saját szervezetek kezelése

Bejelentkezés után a felhasználók számára megjelenik a navigációs sávban a *Saját szervezetek menüpont*, ahol azokat a szervezeteket tekinthetik meg, amelyeknél szervezői pozíciót töltenek be.

Egy kiválasztott saját szervezet oldalán – hasonlóan más szervezetekhez – megjelennek az aktív projektek, azonban a projektkártyákon a feliratkozási és kedvencek közé helyezési ikon mellett megjelenik egy újabb ikon is, amellyel a programok láthatósága állítható. A nyilvános projektek mindenki számára láthatók, míg a rejtett kezdeményezéseket kizárólag a szervezet szervezői tekinthetik meg.

A rejtett projektek mellett a szervezők innen érhetik el a projektlétrehozási űrlapot is. A szervezőknek lehetőségük van a szervezet adatainak módosítására, beleértve a nevet, telefonszámot, e-mail címet, leírást, banki fiók adatokat. Itt frissíthetőek a szervezethez feltöltött képek is, továbbá a szervezők ezen az oldalon adhatnak hozzá új szervezőket a szervezethez. Ehhez egy már regisztrált felhasználó felhasználónevének megadása szükséges.



20. ábra. Értesítések megjelenítése

Értesítések megjelenítése

A 2.1.4. fejezetben részletezett felhasználói értesítések az alkalmazás bármely pontjáról elérhetőek. Ezek megjelenítése az alkalmazás jobb oldalán, állandóan jelen levő lila harang ikonon keresztül történik.

Az ikonra kattintva megjelennek a beérkezett értesítések. Az értesítések esetében, ahogy a 20. ábrán is látható, lehetőség van az értesítést kiváltó projekthez navigálni, vagy egyszerűen megjelölni azokat olvasottként.

Adminisztrátor funkciók

Az adminisztrátoroknak lehetőségük van az oldalon feltüntetett projekt-típusok kezelésére. Ezek a típusok hozzárendelhetőek a kezdeményezésekhez és böngészéskor ezek alapján lehet szűrni azokat.

Egy másik adminisztrátori funkció a szervezetlétrehozási kérelmek elbírálása. Az adminisztrátorok a felhasználók által benyújtott kérelmeket átnézve dönthetnek arról, hogy engedélyezik-e az új szervezet profiljának létrehozását.

Az adminisztrátori funkciók egy kizárólag rendszergazdák számára látható navigációs sávon érhetőek el. Az adminisztrátorok egy ikon segítségével válthatnak az alapértelmezett (mindenki számára látható) és az adminisztrátori navigációs sáv között.

Következtetések és továbbfejlesztési lehetőségek

A fejlesztés során a Watch My Project kitűzött főcéljaihoz tartozó funkcionalitások sikeresen megvalósultak. A webalkalmazás lehetőséget ad arra, hogy bármely civil szervezet vagy intézmény megoszthassa a közösségével és minden egyes érdeklődővel a kezdeményezéseit, így az applikáció egységes, globális gyűjtőhelye ezeknek a projekteknek.

A szervezetek a szavazások, kérdőívek, illetve a kialakított értesítési rendszer segítségével naprakész tájékoztatást nyújthatnak a felhasználóknak. Ugyanakkor a közösség tagjai számára lehetőséget kínálnak arra is, hogy bekapcsolódjanak a megvalósításokba önkéntesség vagy adományozás formájában.

Az alkalmazás felhasználók számára még használhatóbbá, valamint továbbfejlesztés és karbantartás szempontjából fenntarthatóbbá tételéhez többek között a következő továbbfejlesztési lehetőségek körvonalazódtak.

A felhasználói élmény javításának érdekében tervben van a projektek, szervezetek ajánlása földrajzi elhelyezkedés alapján. Az ajánlások a felhasználó eszközének aktuális helyzete és a szervezetek által megadott lokációk alapján történnének.

Az önkéntesek és szervezők hatékonyabb együttműködését elősegíthetné egy applikáción belüli kommunikációs lehetőség bevezetése. Ezáltal a felhasználók közvetlen üzeneteken keresztül beszélhetnének egymással.

A karbantarthatóság és az új funkcionalitások beépítésének gördülékenyebbé tételéhez szükséges automatikus tesztek bevezetése a meglévő kitelepítési folyamatba.

A felsoroltak mellett az egyéb fejlesztési lehetőségek közé tartozik beépített adományozási lehetőség létrehozása, szervezők által megadott információk többnyelvűsítése és az adminisztrátori funkciók kiegészítése.

Hivatkozások

[1] Christopher Cowell, Nicholas Lotz és Chris Timberlake. *Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples*. Packt Publishing Ltd, 2023.

[2] Paul DuBois. *MySQL*. Addison-Wesley, 2013.

[3] Daniel Durante és Sascha Depold. *Supercharging Node. js Applications with Sequelize: Create high-quality Node. js apps effortlessly while interacting with your SQL database*. Packt Publishing, 2022.

[4] *Fetch Standard*. URL: https://fetch.spec.whatwg.org/ (elérés dátuma: 2025. ápr. 3.)

[5] Ian Fette és Alexey Melnikov. *The websocket protocol*. Technikai jelentés. 2011.

[6] *Figma hivatalos oldala*. URL: https://www.figma.com/ (elérés dátuma: 2025. ápr. 7.)

[7] David Flanagan. *JavaScript: the definitive guide*. " O'Reilly Media, Inc.", 2011.

[8] *Folder Structures in React Projects*. URL: https://dev.to/itswillt/folder-structures-in-react-projects-3dp8 (elérés dátuma: 2025. ápr. 3.)

[9] Daniel W Goldberg. „A geocoding best practices guide". (2008).

[10] Jonathan M Hethey. *GitLab repository management*. Packt Publishing, 2013.

[11] *Hivatalos ESLint dokumentáció*. URL: https://eslint.org/docs/latest/ (elérés dátuma: 2025. ápr. 7.)

[12] *Hivatalos Leaflet dokumentáció*. URL: https://leafletjs.com/reference.html (elérés dátuma: 2025. márc. 17.)

[13] *Hivatalos Material UI dokumentáció*. URL: https://mui.com/material-ui/getting-started/ (elérés dátuma: 2025. ápr. 7.)

[14] *Hivatalos React dokumentáció*. URL: https://react.dev (elérés dátuma: 2025. márc. 14.)

[15] *Hivatalos react-i18next dokumentáció*. URL: https://react.i18next.com/ (elérés dátuma: 2025. ápr. 7.)

[16] *Hivatalos TanStack Query dokumentáció*. URL: https://tanstack.com/query/latest/docs/framework/react/overview (elérés dátuma: 2025. ápr. 7.)

[17] *JWT IETF dokumentáció*. URL: https://datatracker.ietf.org/doc/html/rfc7519 (elérés dátuma: 2025. ápr. 12.)

[18] Azat Mardan. *Pro express. js: Master express. js: The node. js framework for your web development*. Springer, 2014.

[19] Azat Mardan, Mardan és Corrigan. *Practical Node. js*. Springer, 2018.

[20] Mark Masse. *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.", 2011.

[21] Ian Miell és Aidan Sayers. *Docker in practice*. Simon és Schuster, 2019.

[22] *MinIO főoldal*. URL: https://min.io/ (elérés dátuma: 2025. ápr. 10.)

[23] *Oauth hivatalos odala*. URL: https://oauth.net/2/ (elérés dátuma: 2025. ápr. 10.)

[24] *Promise-JavaScript*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (elérés dátuma: 2025. ápr. 3.)

[25] *React useContext hook*. URL: https://react.dev/reference/react/useContext (elérés dátuma: 2025. márc. 14.)

[26] *React: Google Login komponens*. URL: https://www.npmjs.com/package/react-google-login (elérés dátuma: 2025. márc. 14.)

[27] Ken Schwaber. *SCRUM Development Process*. Springer London, 1997.

[28] *Sequelize CLI útmutató oldala*. URL: https://sequelize.org/docs/v6/ (elérés dátuma: 2025. ápr. 12.)

[29] Luis de la Torre és mások. „Using server-sent events for event-based control in networked control systems". *IFAC-PapersOnLine* 52.9 (2019), 260–265. oldal.

[30] *Traefik hivatalos oldala*. URL: https://traefik.io/traefik/ (elérés dátuma: 2025. ápr. 12.)